Vector Database infrastructure requirements

white paper

H. Tummala, A. Ahrabian, D. Gerhart, C. Shorb

1. Contents

2. Introduction

2.1 Vector Database

Unstructured data is transformed into a machine-understandable format through a computationally intensive process of converting the unstructured data into vectors — high-dimensional mathematical representations. These vectors capture not only the raw text but also the context and semantic meaning of the content. Metadata can also be extracted from raw data and stored alongside the transformed data.

The importance of vector databases lies in their ability to efficiently store, manage, and index massive quantities of high-dimensional vector data. Unlike traditional relational databases, which organize data in rows and columns, vector databases represent data as vectors. These vectors are clustered based on similarity, enabling low-latency queries for Associated Nearest Neighbors (ANN)—a crucial feature for AI-driven applications.

Vectors are essential because they serve as numerical representations for complex objects, such as words, images, and audio. For instance:

- **Text**: Chatbots rely on vectors to understand natural language. Words, paragraphs, and entire documents are converted into vectors using machine learning algorithms.
- **Images**: Image pixels can be described numerically and combined into high-dimensional vectors.
- **Speech/Audio**: Sound waves can also be represented as vectors, enabling applications like voice recognition.

As organizations handle ever-growing volumes of unstructured data for AI, vector embeddings and vector databases become increasingly important. Vector embeddings represent vectors in a continuous, multi-dimensional space.

Representing unstructured data as a vector embedding allows for similarity searches that traditional databases and data types cannot support, as they require an exact match on the value. In a vector format, not only can we achieve hits for data that is similar or close enough, but there are also various types of calculations that enable different ways of comparing the vectorized data during search time.

Vector databases are crucial in real-time recommendation systems for e-commerce and streaming services, enabling personalized recommendations with low latency. In financial institutions, they aid in fraud detection by analyzing transaction patterns and identifying anomalies in real-time. For platforms relying on visual content, vector databases facilitate efficient image and video searches based on visual features. In natural language processing applications like chatbots, they support semantic search, sentiment analysis, and language translation. In healthcare, they manage and analyze complex medical data, aiding in diagnostics and personalized treatment plans. Lastly, in IoT applications, vector databases handle continuous sensor data influx, supporting real-time data ingestion, processing, and querying for effective device monitoring and control.

Vector databases play a key role in advanced data processing techniques like Retrieval-Augmented Generation (RAG). RAG leverages the strengths of both retrieval-based and generation-based models to enhance the accuracy and relevance of generated responses. In the following section, we will delve into how vector databases underpin the functionality of RAG systems, enabling real-time data retrieval and processing across various applications.

2.2 RAG (Retrieval-Augmented Generation)

Generative AI (GenAI) is revolutionizing business operations by automating tasks, enhancing customer service, and improving decision-making processes. The high cost associated with training foundational models and the need to maintain data privacy, especially for sensitive business information, present significant challenges in delivering GenAI use cases. Retrieval-Augmented Generation (RAG) workflows provide a practical solution for delivery GenAI outcomes in production with high accuracy.

In the realm of machine learning, the RAG methodology offers a dynamic approach to achieving high quality results based on custom and specialized data. Instead of the traditional route of training a tailored model from scratch, RAG enables a foundational model which may have been fine-tuned, to process and understand new, business-specific, and up-to-date data, prior to generating results at inference time. This methodology enhances the quality of the generated output by aligning it closer to the most pertinent, precise, and relevant business data unavailable during the initial training phase. By retrieving and using custom data from authoritative sources at inference time, the output generated reflects the unique characteristics of the custom business data beyond the general knowledge used during training. It also avoids the need to frequently re-train the model using a fine-tuning process.

It is essential that correct and accurate data is found with minimal latency within the RAG workflow. A granular representation of the proprietary and unstructured data is key to RAG's success. An accurate and granular representation of data can be achieved when the data is meticulously organized and indexed, allowing for efficient retrieval and processing.

Vector databases are crucial in this process by offering robust data retrieval capabilities. They enable businesses to leverage pretrained and sometimes fine-tuned GenAI models, which are primarily large language models (LLMs), while ensuring that the responses and outputs are grounded in their proprietary data. This approach reduces the costs associated with GenAI implementation in production and enhances data security by keeping sensitive information on-premises.

Vector databases are essential for the effective implementation of RAG workflows. They provide the necessary infrastructure to ensure accurate, efficient, and secure data retrieval, enhancing the performance and applicability of GenAI in business operations.

2.3 Vector Database in RAG Workflow

Vector databases are essential for generative AI use cases, grounding foundation models with relevant business data, and supporting workflows in the era of GenAI.

Consider the following streamlined workflow which ensures efficient handling of high-dimensional data, enabling accurate retrieval and context-aware responses.

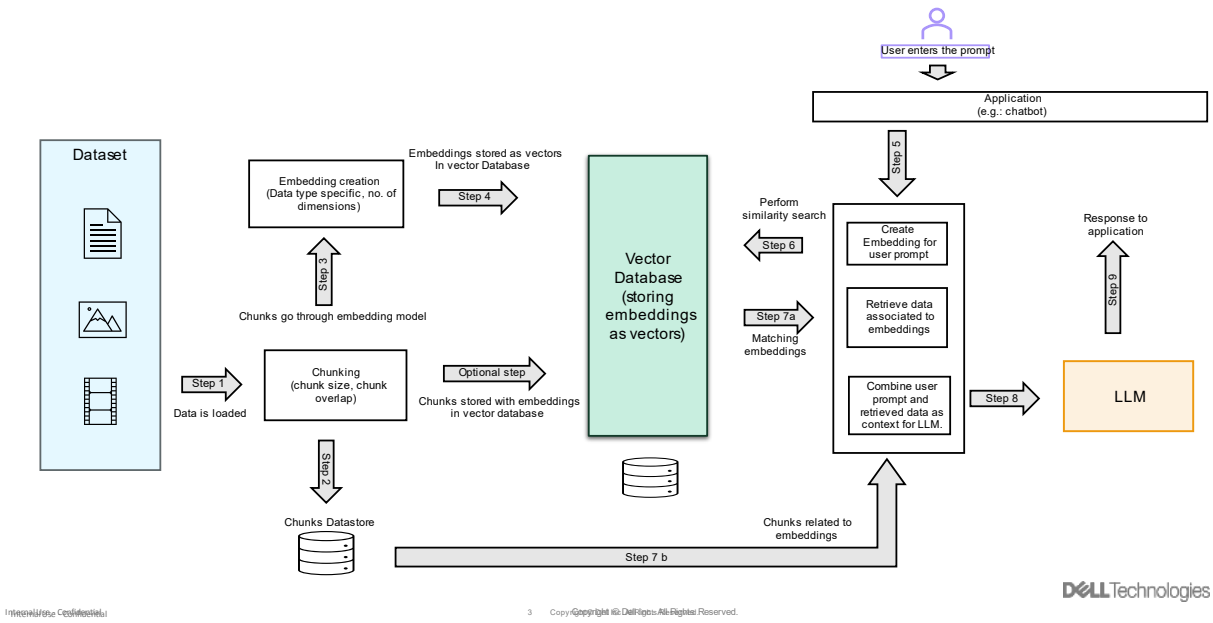


Figure 1: Vector database in a RAG workflow

Figure1 RAG Workflow Steps

1. **Data Set Chunking (step1)**:
    - **Objective**: Break down pre-processed or cleansed large datasets into smaller, manageable pieces (chunks).
    - **Process**: Split copy of datasets documents, text, or other data types into chunks based on predefined criteria (e.g., paragraphs, sentences, or fixed-size tokens). Original dataset (that is the blue box in Figure 1) is untouched during this process.
    - **Outcome**: Each chunk represents a segment of the original data, making it easier to process and analyze. Depending on the chunking strategy, the total size of the chunks may differ from the original data; it could be smaller, equal, or larger.
    -
2. **Creating Embeddings (Step 3)**:
    - **Objective**: Convert each data chunk into a high-dimensional vector representation (embedding).

- **Process**: Use machine learning models, such as pretrained language models, to generate embeddings that capture the semantic meaning and context of each chunk.
- **Outcome**: Each chunk is now represented as a vector, which can be used for similarity searches and other analysis. The vector size depends on the number of dimensions in the embedding and is independent of the chunk size.

3. **Persisting Chunks and Embeddings in Vector Database (Step 2, optional Step, and Step 4)**:
    - **Objective**: Store the chunks and their corresponding embeddings in a vector database.
    - **Process**: Save each chunk along with its embedding in the database, ensuring efficient storage and retrieval.
    - **Outcome**: The vector database now contains a structured representation of the original unstructured data, ready for querying.
4. **Prompt Conversion to Embedding (Step 5)**:
    - **Objective**: Convert a user query or prompt into an embedding.
    - **Process**: Use the same machine learning model to generate an embedding for the prompt, ensuring consistency with the embeddings of the data chunks.
    - **Outcome**: The prompt is now represented as a vector, which can be compared to the stored embeddings.
5. **Retrieving Chunks through Query of Vector Database (Step 6, Step 7a, and Step 7b)**:
    - **Objective**: Find the most relevant data chunks based on the prompt embedding.
    - **Process**: Query the vector database using the prompt embedding to retrieve chunks with similar embeddings (i.e., those that are semantically close to the prompt).
    - **Outcome**: The retrieved chunks are the most relevant pieces of data, providing contextually appropriate responses or information based on the original query.

Summary: This workflow ensures that the generated output is closely aligned with the most pertinent and up-to-date business data, enhancing the quality and relevance of the results.

3. Vector Database Architecture Overview

Vector databases play a critical role in AI workflows by enabling efficient data retrieval and scalability. To meet the performance requirements of AI applications, vector databases must seamlessly scale for data ingestion and query processing.

Key considerations for designing vector database architectures are:

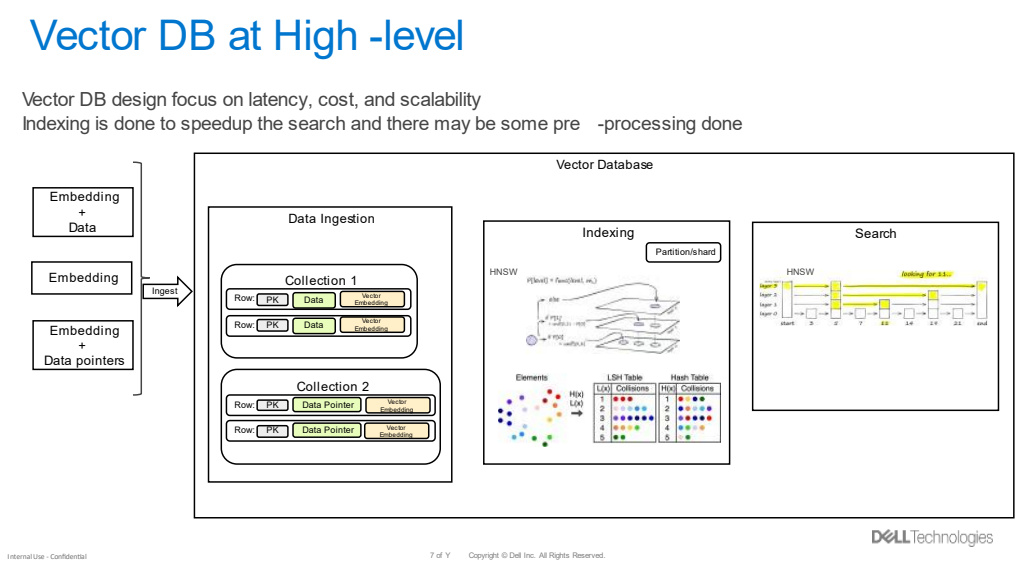


Figure 2: Vector database architecture.

1) Fast Data Retrieval:
    - Semantics Search: Retrieving relevant data quickly is essential for AI applications. Vector databases should excel at semantic-based searches.
    - Low Latency: Performant ingestion, indexing, and searching implementations minimize latency.
2) Modular Architecture:
    - Resource Allocation: Unique features may require varying resources during the data lifecycle. Modular architecture allows independent scaling of specific feature modules as needed.

- Efficient leverage of modules: In a modular architecture, prioritize use of more resource efficient modules while minimizing dependency on resource-intensive modules. This approach streamlines performance, optimizes resource utilizations, and reduces overhead.

3) Common Modules:
    - Load Balancer: Redirect I/O efficiently across nodes.
    - High Availability (HA): Monitor and manage different nodes to ensure system reliability.
    - Query Planner: Optimize query execution plans.
    - Worker Nodes: Perform specific functionality (e.g., data ingestion, indexing, similarity search, and query execution).
4) Compute Utilization:
    - Indexing Module: GPUs accelerate high-dimensional vector operations. Leverage GPUs for efficient indexing.
    - Data Loading Module: CPU suffices for data (embeddings) loading; GPUs are unnecessary. Creation of embeddings is outside of vector database construct.
    - Query Module: Use GPUs for fast similarity search during query execution.
5) Storage Options:
    - Shared Nothing Architecture: Each module has isolated dedicated storage. This architecture minimizes contention for storage resources.
    - Shared Architecture: Modules share storage resources. Optimization is based on I/O latency requirements and throughput.

Summary: Vector databases must strike a balance between performance, scalability, availability, and resource utilization.

4. Vector Database Key Infrastructure

Deployments for vector databases are quite broad:

- Embedded – Smaller vector databases that can be packaged with an application. Examples: SQLite with sqlite-vss extension; Qdrant vector database.

- In-Memory – Small, ephemeral databases that leverage host memory and focus on near-real time analytics. Examples: Redis and SingleStore are in-memory databases with vector data support.
- Monolithic – Databases that operate as a single unified system, often on a single server. They are useful for small to medium-sized databases. They are often traditional single system scale-up architectures and are deployed closer to the source of data generation. Increased performance demand requires scaling-up server resources to meet the increased performance demands. Edge is an increasingly common use case for this type of vector database. Example: PostgreSQL with pgvector extension supporting vector data.
- Microservices/Distributed– Databases that are part of a distributed system. They are often designed as scale-out architecture to scale horizontally and operate as part of an enterprise microservices architecture. Useful for Medium to extremely large databases. Example: Milvus: Milvus pure vector database and Elasticsearch.
- Cloud: Databases that are cloud-based offer scalability and flexibility, integrating with various cloud services and infrastructure. Example: Amazon OpenSearch and Google Cloud SQL with pgvector.

Vector databases have key infrastructure characteristics on which they run; Compute, network, and storage infrastructure are crucial for performance, scalability, high availability (HA), and recovery.

Server Infrastructure:

- Processing Power: High-dimensional vector data requires substantial computational resources. Modern servers with multicore processors and large memory capacity are optimal for efficient vector operations, such as similarity searches and clustering.
- Parallelization: The CPU intensive operations for Index and Query are mostly data independent. This allows Vector databases to benefit from parallel processing. Servers with multiple cores can handle concurrent queries, improving response times.
- GPU Utilization: GPUs can significantly assist with additional computation, memory, and parallelization resources. These additional resources can enhance the performance of vector databases.
- Fast Local Storage: When the size of vector data exceeds system RAM, abundant local disk is key for performance.

Network:

- Latency: Low-latency communication between servers and clients is critical. Vector databases often serve low-latency applications (e.g., recommendation engines), where delays impact user experience.
- Bandwidth: High-dimensional vectors can be large. Adequate network bandwidth ensures smooth data transfer between clients and servers.
- Distributed Systems: Vector database processing may be distributed across multiple servers. A robust network facilitates seamless communication between nodes.

Storage:

- Scalability: Vector databases handle massive amounts of data. Scalable storage solutions (e.g., distributed file systems, object storage) accommodate growth.
- I/O Performance: High-speed storage (SSDs or NVMe drives) reduces read/write latency. Efficient I/O operations are crucial for vector retrieval.
- Compression: Storing high-dimensional vectors efficiently requires specialized compression techniques like quantization which reduce the precision of the vector. Balancing storage size and retrieval speed is essential.

Summary: Optimizing server performance, network responsiveness, and storage scalability is required for vector databases to handle high-dimensional data effectively and support AI workflows and Gen AI applications.

Dell Technologies provides a versatile infrastructure for various vector database architectures, combining high-performance compute, memory, storage, and networking resources. This infrastructure supports scalable and efficient data processing, enabling seamless integration with modern data platforms and AI workloads. Dell's solutions are designed to handle diverse data types and workloads, ensuring optimal performance and flexibility across on-premises, cloud, and hybrid environments.

4.1 Server Infrastructure

Vector Database workloads impact all areas of a server resources: RAM, CPU, GPU acceleration, and local storage.

The foundational infrastructure topics are:

1. Memory/CPU
   The baseline system configuration for a Vector Database is (largely) in-memory with CPU processing. This is the standard server architecture for Vector Database processing unless there is a need for more costly parallel acceleration (i.e., GPU, ...GPUs).

   In addition to classic memory utilization techniques such as caching, compression, pooling, garbage collection, and smart swapping, vector databases will use memory optimized structures (quantization, sparse vectors) and indexes optimized for in-RAM searching.

2. High Speed Local Disks
   As in-memory resources are exhausted vector databases take advantage of disk optimizations for large-scale datasets. These are designed to work efficiently with large-scale datasets by leveraging secondary storage, such as SSDs, or hard drives while maintaining high query performance.

   The common components of this architecture are persistence, data logs, backup, and configuration storage as well as a sophisticated hybrid storage algorithms that maximize storage of frequently accessed data in memory while keeping the rest on disk, aiming for efficient use of memory and storage resources.

3. GPU
   Some vector databases allow GPU acceleration*. This is achieved by offloading aspects of vector search and indexing to GPUs for increased performance. GPU offloading allows computational parallelism, additional memory, increased memory bandwidth, matrix operations, tensor calculations, and optimized floating-point operations. For exceptionally large implementations, multiple GPUs may be utilized independently and/or aggregated together.

Dell's PowerEdge server portfolio is well-suited for vector database computational workflow due to: robust performance, scalability, and reliability. Equipped with the latest Intel® Xeon® processors and high-capacity DDR4 memory, these servers handle the intensive computational demands of vector databases efficiently, ensuring fast data processing and query handling.

For multimode configuration, the following table illustrates the progression of a vector database system as it scales from a single node setup to a multi-node cluster.

Configuration	Single Node	Single Node + GPU	Multi-node Cluster
Server Model	PowerEdge XS	PowerEdge XA	PowerEdge XS(s), PowerEdge XA(s)
Database Type	Standalone VDB	Standalone VDB	Scalable Cluster VDB
Storage	Dell Storage	Dell Storage	Dell Storage

**Note: It is worth highlighting that many other subjects on the periphery of vector databases that benefit from GPU acceleration are excluded from this discussion. Example: creation of Embeddings benefits from GPU but this is not covered in the document; embedding generation happens outside of and independently of the VDB.*

4.2 Storage Infrastructure

During the data ingestion phase, achieving fast client response time is essential. Traditional data persistence architectures face challenges related to network latency, storage requirements, and data protection.

Vector DB Storage Architecture

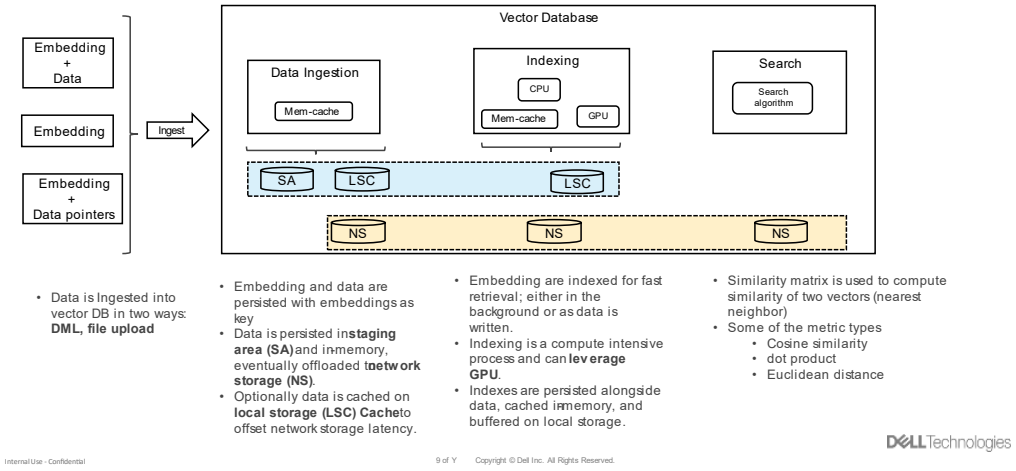


Figure 3 Vector Database Storage architecture.

Challenges

1. Local Storage and Network Latency
    - Issue: Leveraging local storage minimizes network latency during data ingestion. This approach demands significant storage per node.

- Solution: To scale storage independently of computational resources, we introduce tiered storage, where low-access data (cold data) resides in network-attached storage (NAS) or S3 storage.

2. Data Resilience
    - Issue: Ensuring data protection requires replication which exacerbates storage demands.
    - Solution: By tiering data, we reduce the reliance on local storage, mitigating the impact of replication.
3. Tiered Storage Approach
    a. Cold Data Tier
        - Definition: Cold data refers to infrequently accessed data.
        - Storage Location: Store cold data in network storage (NAS or S3).
        - Benefits:
            - Efficiently manage large volumes of data without overwhelming local storage.
            - Scale storage independently of computational resources.
            - Reduce the need for extensive replication.
    b. Warm Data Caching
        - Definition: Warm data includes frequently accessed data during query and indexing.
        - Caching Strategy
            - Pre-fetch warm data from network storage to local storage.
            - Cache frequently accessed vectors for faster retrieval.
            - Evict or update cached entries based on access patterns.
4. Impact on Application Response
    a. Latency Reduction
        - Query Time: Fetching data from network storage introduces latency.
        - Optimization: Caching warm data locally minimizes query time latency.
    b. Indexing Efficiency
        - Index Creation: Indexing benefits from local storage.
        - Trade-off: Balancing storage requirements with indexing performance.

Note: Limited RAM requires usage of high speed-local disk. Limited local disk requires usage of external storage. Larger databases/datasets require efficient storage at all resource levels.

Dell's Power family of storage solutions is designed to meet the requirements of vector databases, ensuring optimal performance, scalability, and reliability.

- Dell PowerStore is designed to handle high-performance transactional workloads efficiently. It offers advanced data reduction capabilities, intelligent automation, and scalable architecture.
- Dell PowerScale provides a robust solution for scale-out workloads, particularly those involving substantial amounts of unstructured data. It supports both file and object storage interfaces, making it suitable for applications like Milvus that require scalable and flexible storage solutions.
- Dell PowerFlex is a comprehensive software-defined infrastructure platform that provides independent scaling of compute and storage. It offers flexibility in deployment, extensive automation, and scalable architecture to meet the demands of modern workloads. PowerFlex delivers high performance with sub-millisecond latency, linear scalability, and guaranteed data reduction capabilities.

5. Vector Database Scaling, Indexing, and Querying

5.1 Data Partitioning and Sharding

Sharding is a technique used in database management systems to partition data across multiple servers or nodes. This approach is particularly useful for improving scalability and performance by distributing the workload and storage requirements.

Milvus uses sharding to handle large~~-scale~~ datasets efficiently. Milvus employs sharding to divide embeddings (vector data) across multiple nodes, where each node is responsible for storing and processing a subset of the vectors. This allows for horizontal scaling of data ingestion even with large amounts of data and for high-dimensional vectors. The separation of data further allows horizontal scaling of downstream Index and Query processing.

PostgreSQL uses sharding to partition tables and distribute them across multiple physical or logical nodes. Each shard in PostgreSQL can be managed independently, allowing for better distribution of read and write operations. This scalability enhancement is crucial for applications that require handling large volumes of data without sacrificing ACID compliance and while maintaining low-latency access and high availability.

5.2 Indexing

Creating indexes on the ingested data facilitates fast retrieval. Optimizations include choosing an appropriate index type based on the dataset size and query requirements, and tuning index parameters to balance accuracy and performance.

Vector database indexing methods include IVFFLAT, HNSW, DiskANN, GPU_CAGRA, and GPU_IVF_FLAT.

1. IVFFLAT uses the Inverted File technique and the FLAT library for approximate nearest neighbor searches, balancing search accuracy with computational efficiency.
2. HNSW is designed for efficient nearest neighbor search in high-dimensional spaces, known for its fast and scalable retrieval of similar vectors.
3. DiskANN is optimized for scenarios where data resides on disk rather than in memory, making it suitable for datasets larger than available memory. While slower than in-memory indexing, it is necessary for large datasets. The lower memory requirements can potentially be a lower cost alternative to high-memory systems.
4. GPU_CAGRA is a graph-based index optimized for GPUs, using inference-grade GPUs for cost-effective performance. It is based on CAGRA, RAFT's new state-of-the-art ANN index. It is a high-performance, GPU-accelerated, graph-based method that has been specifically optimized for small-batch cases, where each lookup contains only one or a few query vectors.
5. GPU_IVF_FLAT is like IVF_FLAT, where vector data is divided into 'nlist' cluster units. Vector comparison is made between the target input vector and the centroid of each cluster; optimized for GPU acceleration.

The type of computational method used affects indexing performance and cost.

1. CPU-based indexing is generally slower than GPU-based indexing, especially for large-scale datasets, with CPU-based Indexing generally having a lower initial cost. It is suitable for smaller datasets or applications where real-time performance is not critical.
2. GPU-based indexing is significantly faster due to the parallel processing capabilities of GPUs, making it ideal for handling large-scale and high-dimensional data. Although it has a higher initial cost due to expensive GPU hardware, it can be more cost-effective overall for applications requiring high performance.

Overall, the choice of indexing method and computational resource type significantly impacts the performance and cost of vector databases. CPU-based methods are cost-

effective for smaller datasets, while GPU-based methods offer superior performance for larger scale applications.

5.3 Query Process

This stage involves processing the query to retrieve relevant vectors from the database. Several optimizations can be applied to improve performance:

- **Parallel Query Execution**: Running multiple queries in parallel to utilize available computational resources effectively. This can significantly reduce query response times.
- **Query Caching**: Storing the results of frequently executed queries in a cache. This allows for quick retrieval of results without re-executing the entire query, reducing computation time.
- **Load Balancing:** Distributing query loads across multiple nodes or servers to prevent bottlenecks and ensure consistent performance. This helps in handling high query volumes and maintaining system stability.

Optimizing query processing involves selecting appropriate techniques and compute resources to balance performance, accuracy, and cost. By fine-tuning each stage of query processing, vector databases can deliver fast and accurate query results, even for large and complex datasets. CPU-based methods are cost-effective for smaller datasets, while GPU-based methods offer superior performance for large-scale and real-time applications. Disk-based indexing is essential for handling datasets that cannot fit in memory.

6. Vector database Storage Optimizations

Vector databases often use compression techniques to reduce the memory footprint and improve the efficiency of vector similarity searches. These techniques typically involve reducing the precision of vector elements or transforming the data into a more compact representation.

List of some of Compression Techniques

1. Quantization
    - Binary Quantization (BQ): Converts vectors into binary codes to reduce storage requirements.

    - Product Quantization (PQ): Divides vectors into smaller sub-vectors and quantizes each sub-vector separately.
    - Scalar Quantization (SQ): Reduces the precision of each vector element, often by converting floating-point numbers to integers.
2. Dimensionality Reduction
    - Principal Component Analysis (PCA): Reduces the number of dimensions by transforming the data into a new coordinate system where the greatest variances are captured in the first few dimensions.
    - t-Distributed Stochastic Neighbor Embedding (t-SNE): Reduces dimensions while preserving the local structure of the data.
3. Vector Quantization
    - Codebook Generation: Creates a set of representative vectors (codebook) and encodes the original vectors using these representatives.
    - Feature Engineering: Transforms the original vectors into a lower-dimensional space using engineered features.

These techniques help manage the large size and high dimensionality of vector data, making vector databases more efficient and scalable.

7. Workloads and Benchmarks

7.1 Vector database Workloads

Low latency is critical for achieving Return on Investment (ROI) in RAG-based applications. Having the appropriate computational power for various operations is essential for achieving the required performance at the solution level.

In this paper's context, the focus of vector databases in the RAG pipeline may be divided into three distinct stages: Data Ingest, Query Processing, and Response Generation. Vector databases serve as the fundamental component for the Ingest and Query stages.

1. Data Ingest involves the initial input of unstructured data into the vector database. Robust computational power is required to handle large volumes of data efficiently, and to ensure the data is ingested quickly and accurately.
2. Query Processing focuses on retrieving and processing data based on user queries. It demands high performance and low latency to ensure that queries are executed swiftly to provide timely and relevant results.

3. Response Generation is not directly handled by the vector database. This stage involves generating responses based on the processed queries. The efficiency of the vector database in the ingest and query processing stages directly impacts the speed and accuracy of response generation.

For optimized design and sizing of your vector database infrastructure, consider the following factors:

1. Data Size: The total amount of unstructured data.
2. Data complexity and Dimensionality: How complex the data content is and how dense the vector embedding will be. Larger dimensionality of vector data will require more resources. Another factor is whether the vector embeddings are dense or sparse. Another complexity is how specific the system's response should be. The more specific a required outcome on complex data, the more computational resources will be required.
3. Data Updates: Frequency and volume of data changes. Distinguishing between incremental additions and modifications to existing data helps estimate the reindexing load on the database.
4. Data Currency Requirements: Consider the expected speed at which new data must be available for query processing. A vector database must efficiently manage simultaneous data ingestion, re-indexing a, and query processing in use cases where frequent data updates and rapid query responsiveness is essential.

While not exhaustive, the above metrics collectively determine the throughput demands for the data retrieval component of the solution and guide the allocation of computational resources across operational stages.

Workloads can be categorized based on data set size:

1. Small Data Sets: For instance, 500K vectors with 1,536 dimensions.

   Data Ingest: For smaller data sets, the ingest workload is relatively light. This requires less computational power, less storage capacity, and all vector data is in-memory. The focus is on efficiently handling incremental additions and modifications.

   Query Processing: Query processing for small data sets is typically faster and requires fewer resources. The system can quickly retrieve and process data, ensuring low latency responses.

2. Medium Data Sets: For instance, 10 M vectors with 768 dimensions.

Data Ingest: Medium-sized data sets require more robust computational resources to manage the increased volume of data. All vector data is in-memory, requiring sufficient RAM for large monolithic systems, or sufficient nodes for distributed system. Efficient data ingestion and indexing are critical to maintain performance.

Query Processing: With medium data sets, query processing becomes more complex and resource intensive. The system needs to balance speed and accuracy to provide timely responses.

3. Large Data Sets: For instance, 100 M vectors with 768 dimensions

   Data Ingest: Ingesting large data sets demands significant computational power and storage capacity. All vector data cannot reside in memory for monolithic systems. Distributed systems may (or may not) scale to accommodate the vector data processing in-memory. The system must handle high data velocity and ensure that new data is quickly available to query nodes.

   Query Processing: Query processing for large data sets is highly resource intensive. The system must efficiently manage large-scale data retrieval and processing to maintain low latency and high performance.

These categories help in understanding the varying computational and storage requirements based on the size of the data set, ensuring that the vector database infrastructure is optimized for different workloads.

7.2 Vector database – pgvector and Milvus.

Two notable solutions in this space are pgvector and Milvus, each offering unique capabilities to address the challenges of vector similarity search and storage.

7.2.1 pgvector

pgvector is an open-source extension for PostgreSQL that allows storing and searching vector data alongside traditional structured data. This makes it a valuable tool for applications requiring efficient vector similarity searches within a relational database environment.

Here are some key aspects of pgvector:

- **Integration with PostgreSQL**: pgvector is built as an extension to PostgreSQL, leveraging its robust database capabilities.

- **Vector Storage**: It enables the storage of vector data, which is useful for applications like machine learning and similarity searches.
- **Search Capabilities**: Supports both exact and approximate nearest neighbor searches using algorithms like l2_distance, cosine-distance, and vector-negative-inner-product.
- **Indexing**: Implements HNSW and IVFFlat indexing methods to optimize search performance.
- **PostgreSQL Features**: Inherits PostgreSQL's features like Write-Ahead Logging (WAL), replication, and point-in-time recovery, ensuring data integrity and reliability.
- **Memory Requirements**: Search performance depends on the frequently accessed data/index fitting in memory, but not all queried table indexes need to be in memory.

Summary: pgvector enhances PostgreSQL by adding advanced vector search capabilities, making it a powerful tool for modern data applications.

7.2.2 Milvus

Milvus is an open-source vector database designed for high-performance similarity search on large-scale datasets. It supports various deployment options, including standalone, distributed, and cloud-based solutions. Milvus is a valuable tool for applications requiring efficient vector similarity searches within a pure vector database environment.

Here are some key aspects of Milvus:

- **Purpose:** Milvus is built to store, index, and manage massive embedding vectors generated by machine learning models.
- **Scalability:** It supports horizontal scaling, allowing it to handle billions of vectors efficiently.
- **Performance:** Milvus uses advanced indexing techniques and a distributed architecture to ensure fast and accurate similarity searches.
- **Deployment Options:** Offers various deployment models, including standalone, distributed, and cloud-native options, making it flexible for different use cases.
- **Integration:** Works well with AI and machine learning tools, making it a powerful choice for applications like recommendation systems, image retrieval, and more.

Summary: Milvus is designed to make unstructured data search more accessible and efficient, providing a consistent user experience across different environments.

7.3 Benchmark Tools

The open-source tool **VectorDBBench** was selected to compare performance of different vector databases due to its broad support for modern Vector Database platforms and associated index types.

VectorDBBench reports four key performance metrics for each test run:

1. **Load Duration**: Time (in seconds) for data ingestion from the benchmark test driver to the vector database, including post-insert optimization and indexing.
2. **Query / Sec (QPS)**: Average total rate of vector retrieval during the query test phase, with client instances querying the database for 30 seconds at scale (1, 5, 10, 15, 20, 25, 30, 35 clients).
3. **Latency p99**: The 99th percentile latency (ms) of the query requests.
4. **Recall (%)**: Ratio of relevant items retrieved by a query to the total number of relevant items that could have been retrieved, indicating the system's ability to retrieve all relevant items from a dataset.

Performance evaluation was conducted using five index types to organize the vector embeddings:

1. **IVFFLAT**: Utilizes the Inverted File technique and the FLAT library for approximate nearest neighbor searches in large-scale vector datasets. Default parameters: 1024 nlist (number of partitions), 64 nprobe (how many partitions to search for ANN).
2. **HNSW (Hierarchical Navigable Small World)**: Designed for efficient nearest neighbor search in high-dimensional spaces. Default parameters: 30 M (the number of neighbors in the core graph), 360 efConstruction (how many candidate neighbors are considered when adding a new node), 100 ef (size of the dynamic candidate search list).
3. **DiskANN (Disk-Accelerated Nearest Neighbor)**: Optimized for scenarios where data resides on disk rather than in memory. Default parameter: 100 searches_list (number of lists/partitions examined during search).
4. **GPU_CAGRA**: A graph-based index optimized for GPUs, using inference-grade GPUs for cost-effective performance.
5. **GPU_IVF_FLAT**: Similar to IVF_FLAT optimized for GPU.

The following performance tests were conducted:

1. **Search Performance Test (500K Dataset, 1536 Dimensions)**: Tested search performance with a medium 500K dataset (OpenAI 500K vectors, 1536 dimensions) at varying parallel levels.
2. **Search Performance Test (10M Dataset, 768 Dimensions)**: Tested search performance with a large 10M dataset (LAION 10M vectors, 768 dimensions) at varying parallel levels.

8. Results

A vector database benchmark measures the unique efficiency of the database in utilizing compute, memory, and storage resources for vector query processing. It also assesses the scalability of the database from a single node to a multi-node deployment. Additionally, the benchmark provides insights into network and storage configuration impacts during distinct stages of vector data loading, indexing, and querying.

The measured performance results are an indication of this basic principle: The greater amount of computational resources (CPU, GPU) or memory, the better the overall performance.

8.1 Standalone Milvus – Compute

Standalone configuration focuses on a bare-metal installation and the difference between CPU and GPU computational resources. The accelerations used in this test are GPU_CAGRA, GPU_IVF_FLAT which leverages Nvidia rapids raft cuVS CAGRA GPU acceleration algorithm.

Configuration details in Setup Info section.

Note: During the test runs, data is loaded the same way so that the performance measurements relate to the computation of selected indexing scheme.

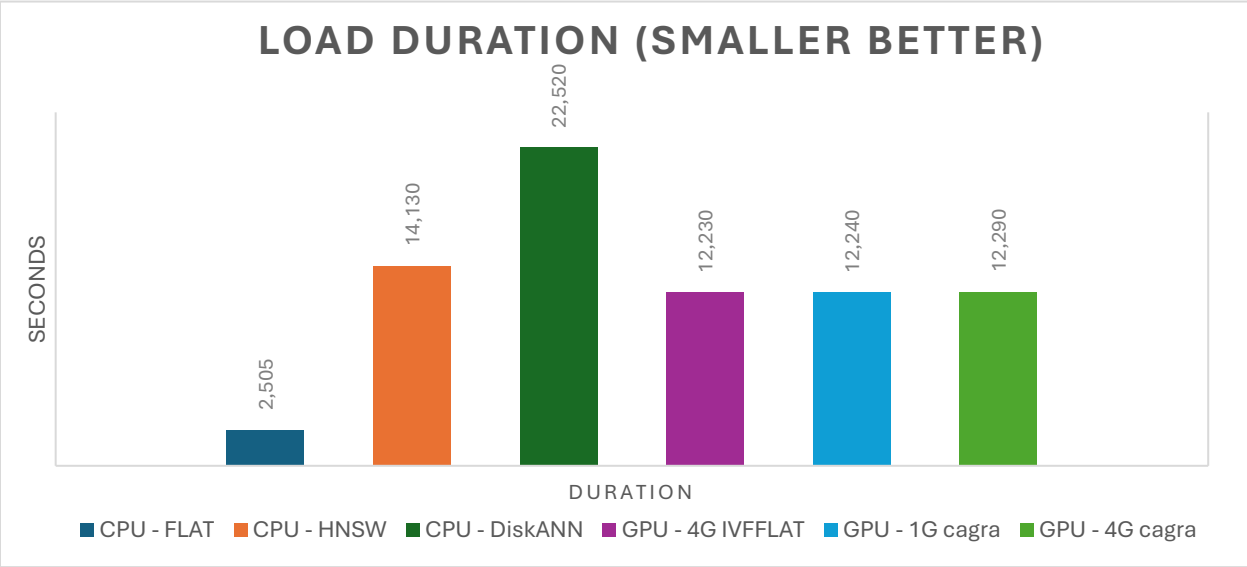


Figure 4 Standalone Milvus Load duration compute test results.

If we try to analyze the combined data, we can make the following general observations:

- Simple flat is the quickest load time as there is no index creation. All vector embeddings are searched for all queries.
- Conversely DiskANN has the largest load time. This suggests it is one of the algorithmically hardest indexes and requires highest CPU processing without any GPU assist.

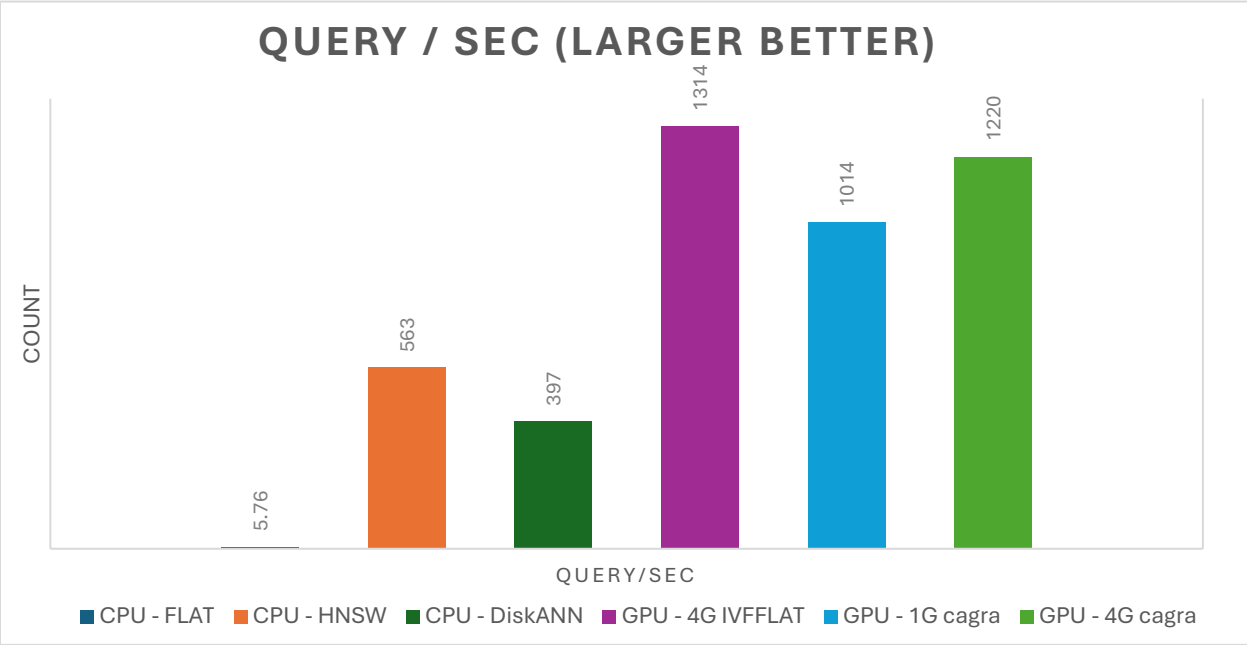


Figure 5 Standalone Milvus Query stage compute test results.

From above results the following can be concluded:

- FLAT - Flat indexing involves comparing the query vector to every single vector in the database. Slow!!
- HNSW – This is an optimized fully in-memory index which performs best among the non-GPU accelerated indexing schemes.
- DiskANN Optimized for disk access, DiskANN will not be as performant for smaller datasets that are entirely in-memory. The reverse would be true for datasets that do not fit into RAM. This highlights the need to map the appropriate index type to the available resources to maximize performance.
- GPU(s) – No surprise: More GPUs improve query performance. The interesting insight is that FLAT maps well and is performant on the GPU parallel architecture.

8.2 Single-node pgvector – VM Hosted

For the most used index mechanism HNSW, the load duration increases with the size of the dataset. As pgvector does data insert and index building in sequence in this test setup, we can break down “load-duration” into Insert processing and Index creation sections. Results indicate insert duration is a bit constant and index building varies with data set

size. Index building time can be optimized by tuning maintenance_work_mem and max_parallel_workers parameters of PostgreSQL.

pgvector does not support DiskANN vector indexing method.

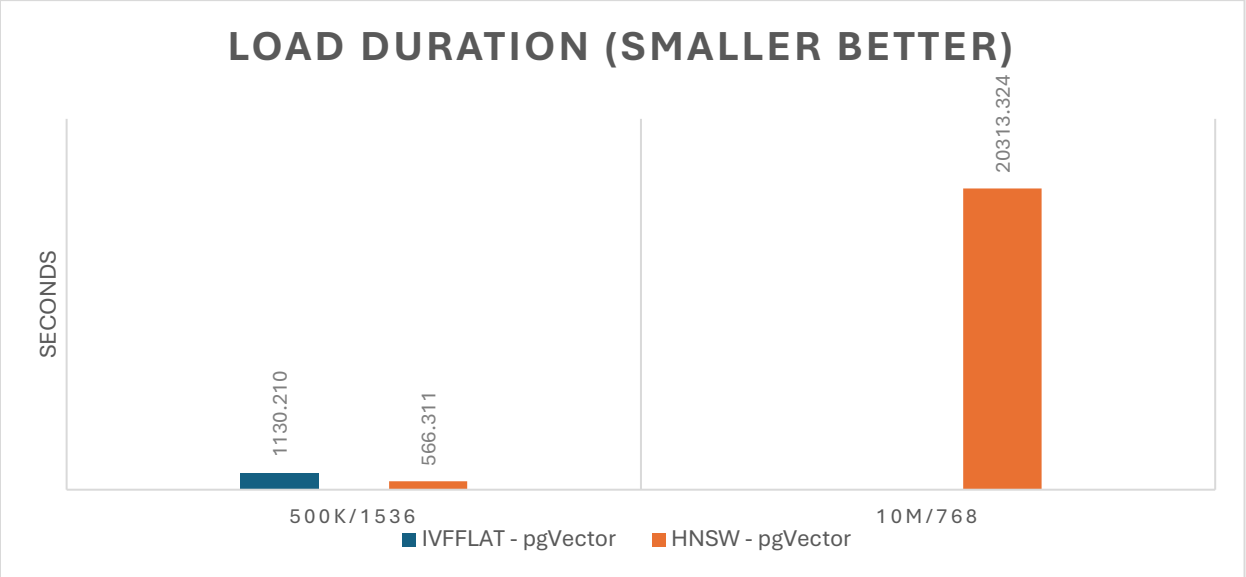


Figure 6 single node pgvector VectorDBBench test results - load duration.

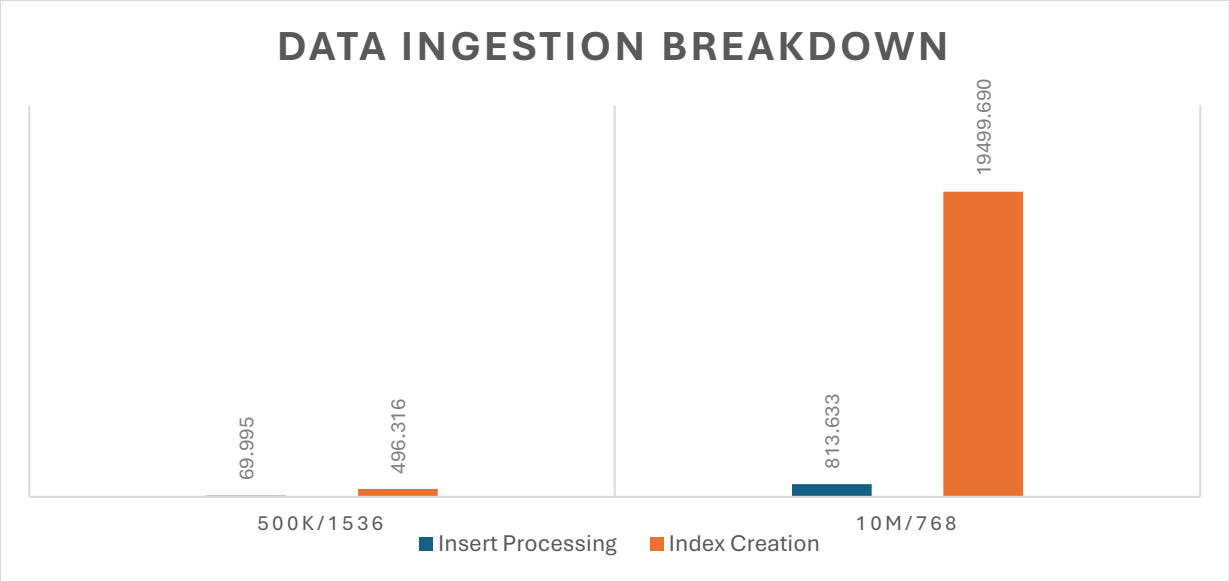


Figure 7 single node pgvector VectorDBBench test results - load duration breakdown.

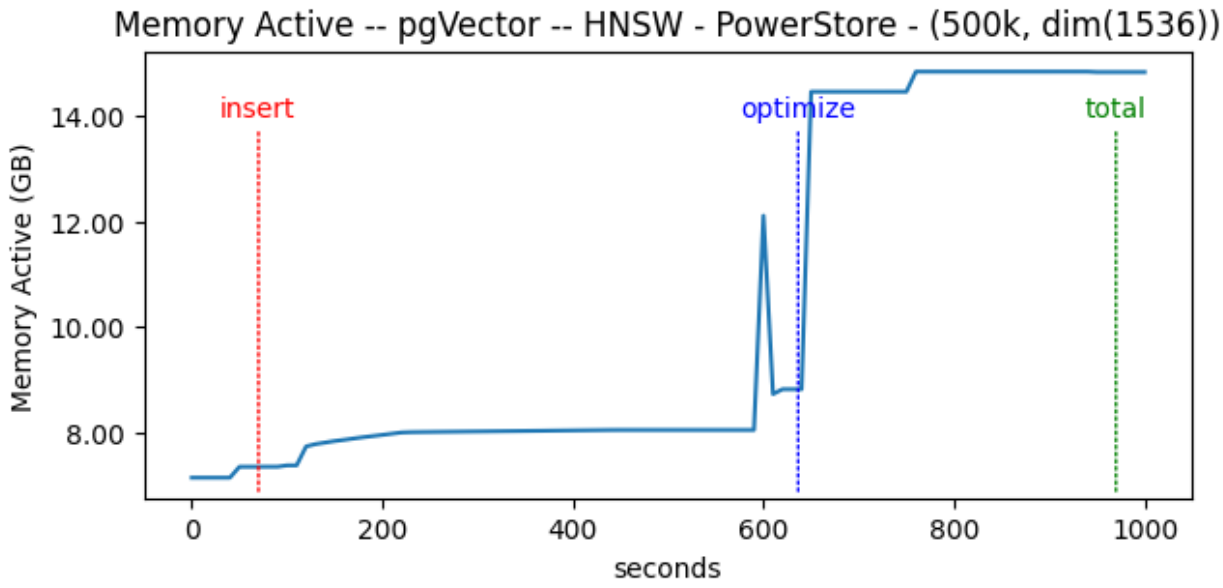


Figure 8 single node pgvector VMstats during vectorDBBench test run - Memory.

As graph in Figure 8 indicates, memory usage increases during index building. All the HNSW indexes stay in-memory, and this in-memory index data is used during query execution.

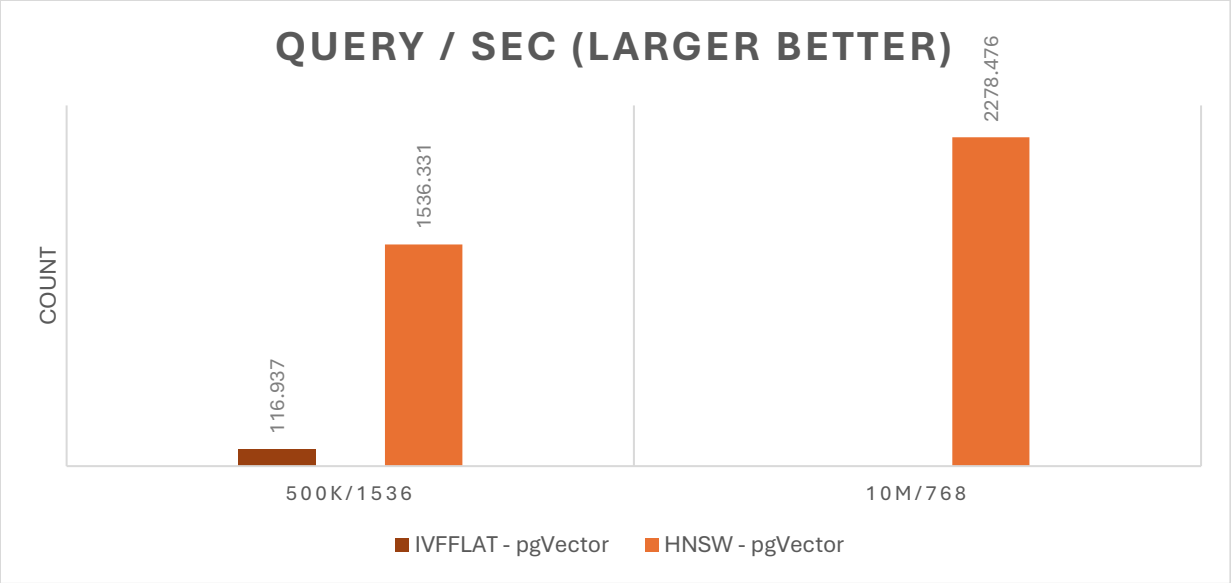


Figure 9 single node pgvector vectorDBBench test run - Query stage.

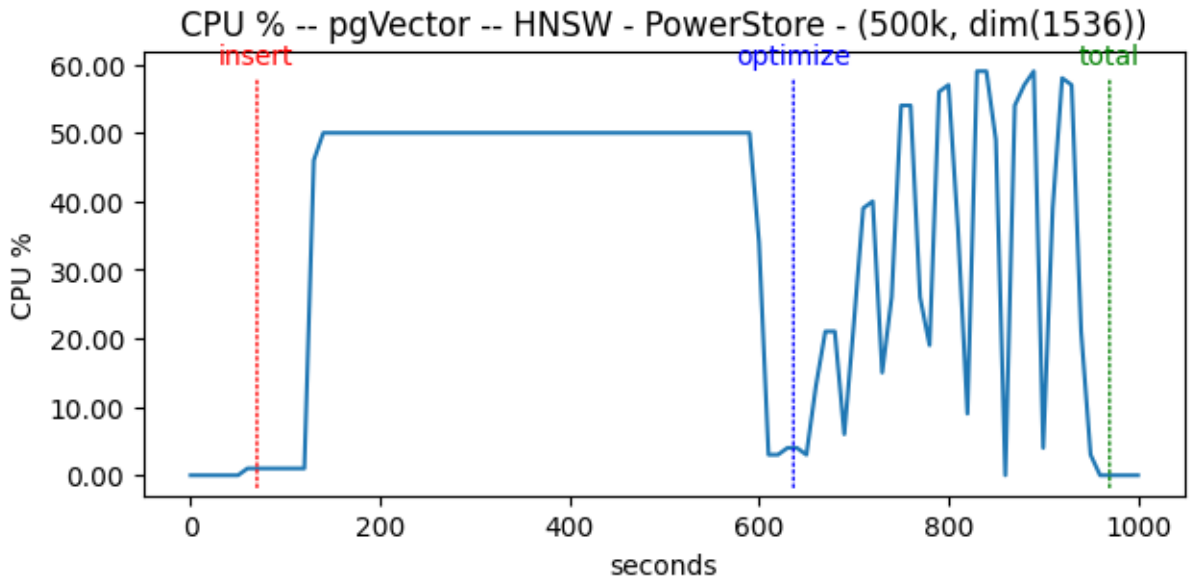


Figure 10 single node pgvector VMstats during vectorDBBench test run - CPU.

Compute resource usage is high during the index building and query stages.

HNSW offers faster query performance, providing quicker responses, which is ideal for high-throughput systems.

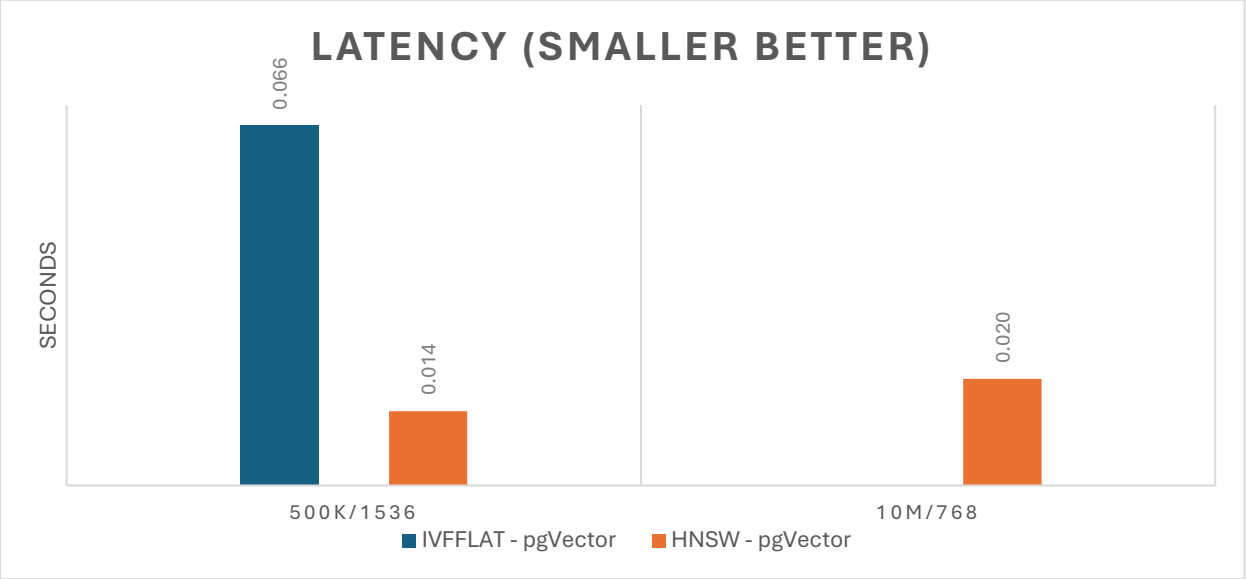


Figure 11single node pgvector vectorDBBench test run - Latency.

IVFFlat is slower in search speed compared to HNSW, but it is ideal for fast index creation and efficient memory usage.

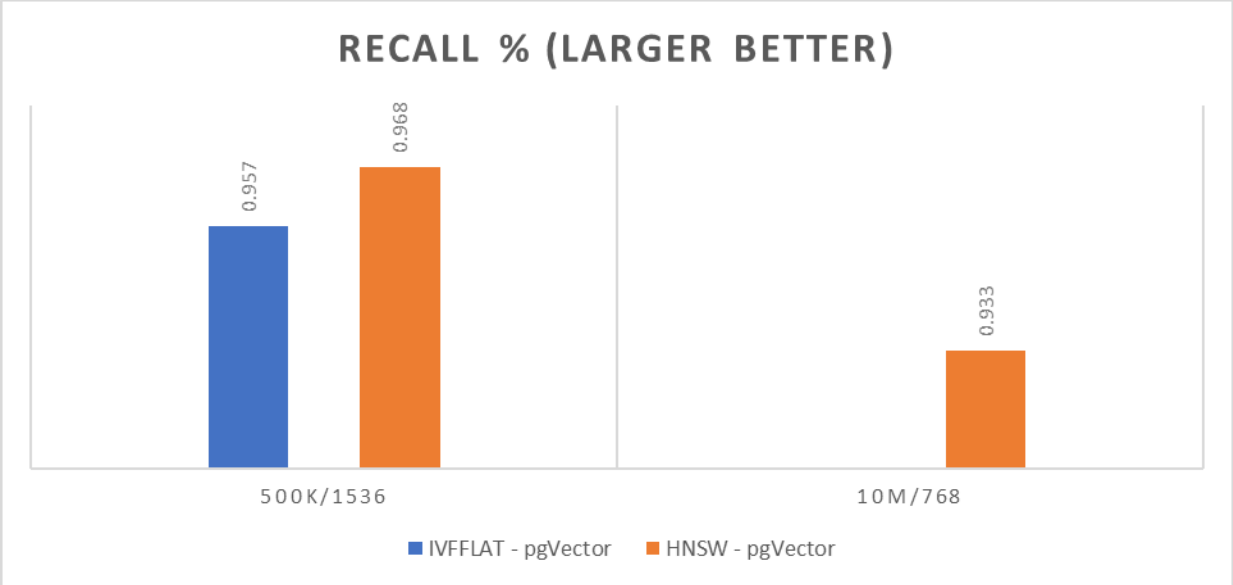


Figure 12 single node pgvector vectorDBBench test run - Recall.

HNSW is more memory-intensive but delivers superior accuracy and recall.

8.3 Three-node Milvus – K8S Hosted

Comparing Milvus performance shows the expected increase in index creation time based on the complexity of the index.

The larger data set shows the advantage of the Milvus distributed architecture.

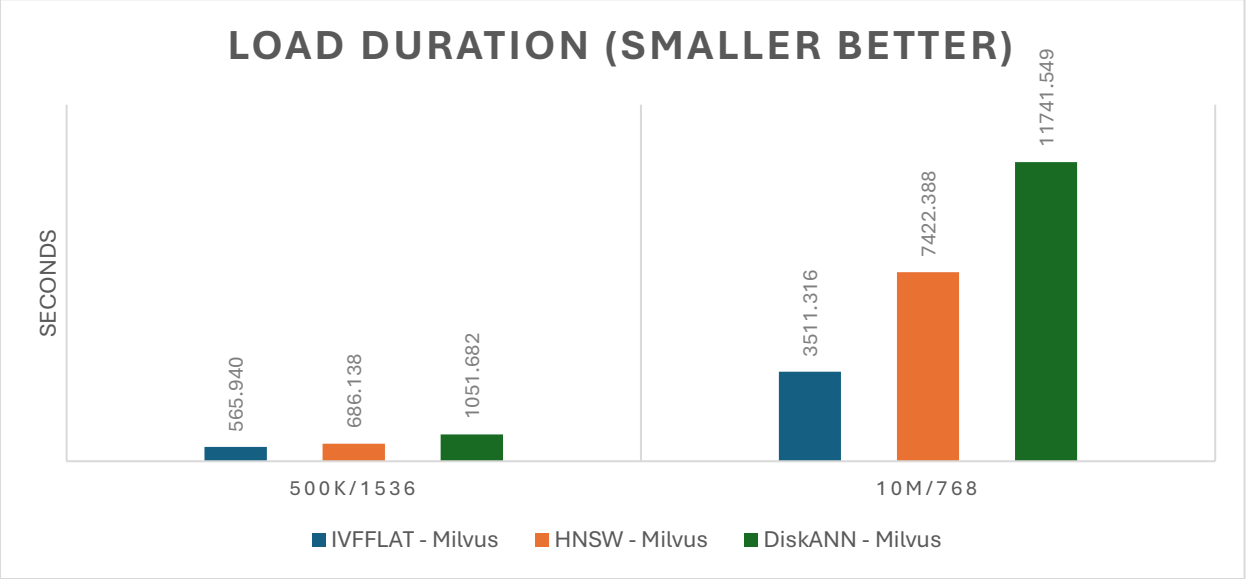


Figure 13 Three-node Milvus vectorDBBench test run - Load duration.

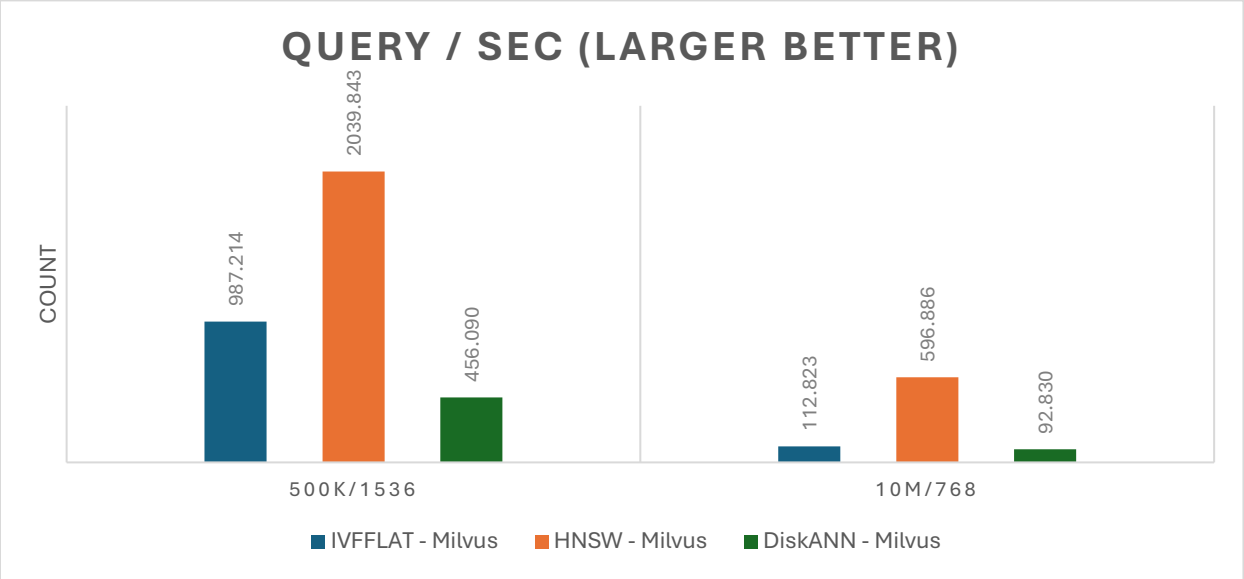


Figure 14 Three-node Milvus vectorDBBench test run - Query.

The index query performance for the Milvus platform has predictable results. The more complex HNSW index performs very well against the more brute force IVFFLAT. As the size of data increased so did the performance delta between them. DiskANN is specifically designed for efficiency in configurations whose data size is greater than available memory. It relies on optimized SSD access in place of all in-memory implementations. The requirement of disk access is seen in lower performance scores.

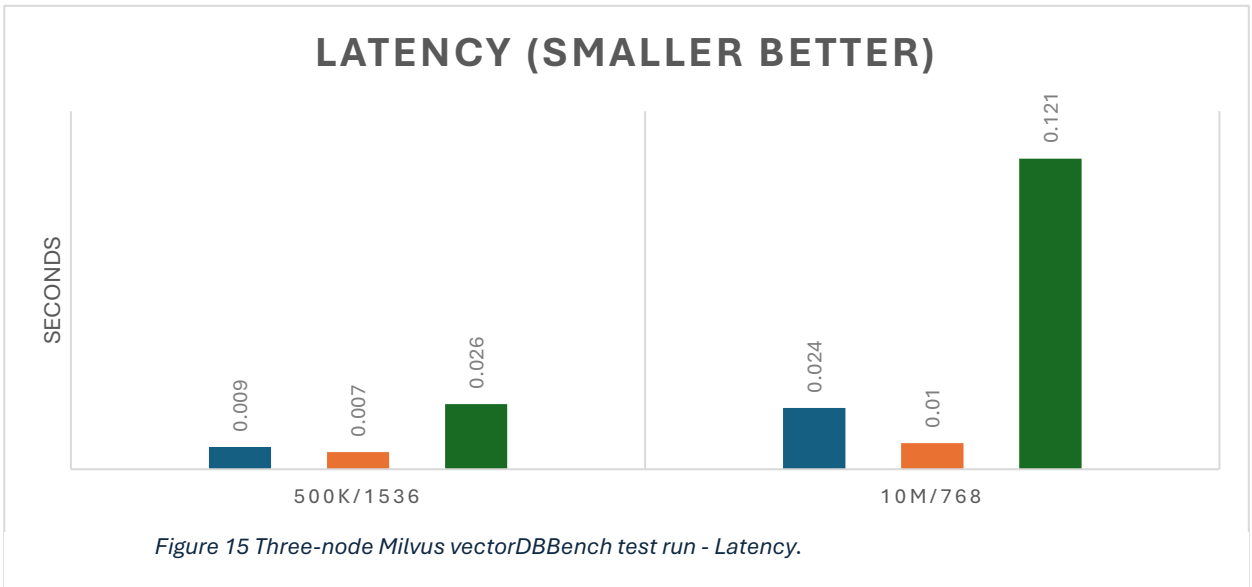


Figure 15 Three-node Milvus vectorDBBench test run - Latency.

As expected, latency relates to the amount of the index that can be loaded into memory. The more disk accesses during processing, the higher the latency.

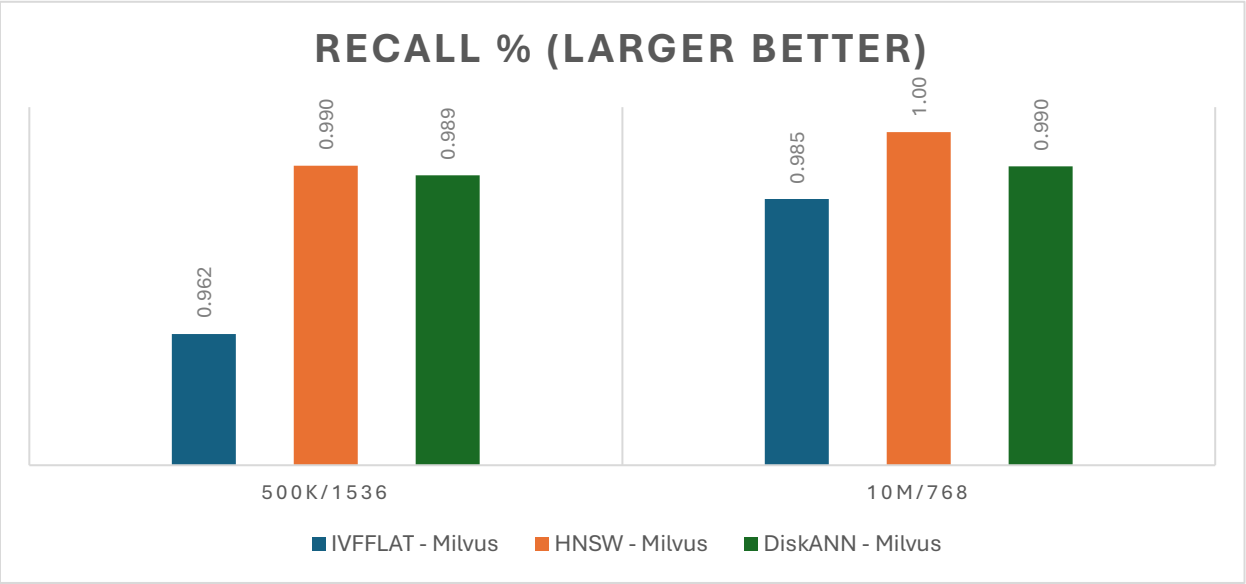
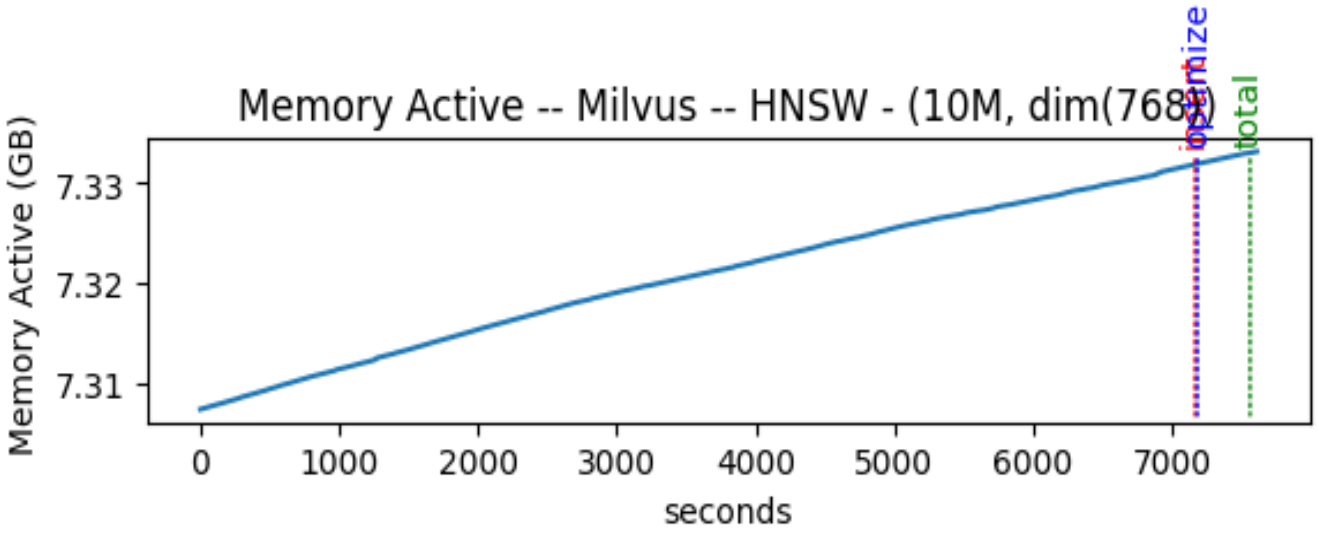


Figure 16 Three-node Milvus vectorDBBench test run - Recall.

The recall rates show no surprises between the various setups.

To understand the system resource usage collected VMstats during the test run. Below graph Figure 17 HNSW memory usage increases as index is built and resides in memory and stays consistent during the query operations.

Figure 17 Three-node Milvus VMStats during vectorDBBench test run - Memory.



Below graph figure Compute resource of usage is high during the index building and query stage as these are the most computational operation intensive stages.

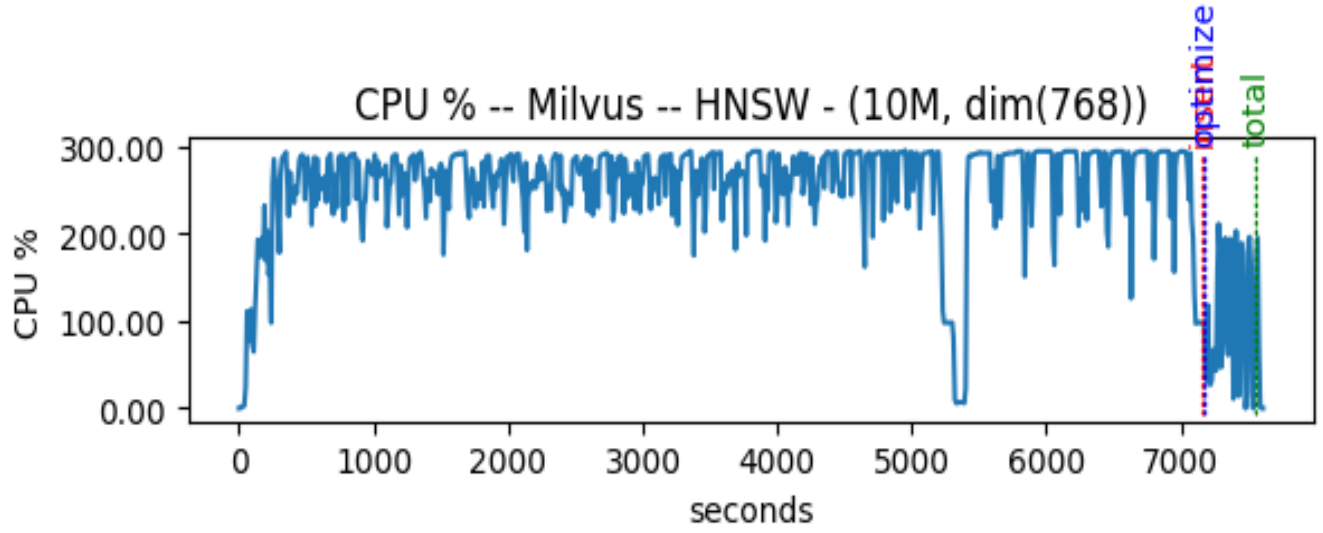


Figure 18 Three-node Milvus VMStats during vectorDBBench test run - CPU.

9. Backup

Given the critical nature of the data stored, robust backup and recovery mechanisms are essential to protect against data loss due to hardware failures, software glitches, or human errors.

The following sections provide high-level summary and references to PostgreSQL backup and restore supported by Dell portfolio and Milvus database backup and restore features.

9.1 Dell PowerProtect Data Manager (PPDM) – PostgreSQL Database

This involves several key steps to ensure application consistent, data integrity and availability. Here is a summary:

Setup and Configuration:

- Install PPDM: Ensure that Dell PowerProtect Data Manager is properly installed and configured in your environment1.
- Configure PostgreSQL: Enable Write-Ahead Logging (WAL) and set up archive logging in the postgresql.conf file to ensure continuous data protection.

Backup Process:

- Full Backups: Schedule regular full backups of the PostgreSQL database using PPDM. This involves creating a complete copy of the database at a specific point in time.
- Incremental Backups: Configure incremental backups to capture only the changes made since the last full backup. This helps in reducing storage requirements and backup time. Except K8s deployment, where CSI only supports full snapshot and hence will always be full backup.
- Application consistency: Full and incremental backup process requires the application to ensure all the application data is flushed to storage. This is advantageous for fast application recovery and data consistency.

Snapshot Management:

- Snapshot Initiation:
    - Based on the defined policies, PPDM initiates the creation of snapshots. This can be done manually or automatically according to the schedule set in the policy.
- Snapshot Creation:

    - PPDM interacts with the underlying storage infrastructure like PowerStore to create snapshots. This process captures the data's state at a specific point in time without disrupting ongoing operations.
- Snapshot Management:
    - PPDM manages the lifecycle of snapshots, including retention and deletion. Policies ensure that snapshots are retained for the required duration and are deleted when they are no longer needed.

Dell PowerProtect Data Manager (PPDM) integrates with PostgreSQL's pg_backup_start and pg_backup_stop functions to ensure consistent and reliable backups.

Recovery Process:

- Restore from Snapshots: In case of data loss or corruption, application admin can restore the database from the most recent snapshot using PPDM. Ensure that the WAL logs are also restored to maintain data consistency.
- Verify Data Integrity: After restoration, verify the integrity of the restored data to ensure that the database is fully operational and consistent.

Monitoring and Maintenance:

- Regular Monitoring: Continuously monitor the backup and recovery processes to ensure they are running smoothly. Use PPDM's monitoring tools to track backup status and performance.
- Maintenance: With PPDM there is no need for regular maintenance tasks such as cleaning up old backups and optimizing storage usage to ensure efficient backup operations.

For Additional references:

- Dell Technologies White Paper: https://www.delltechnologies.com/asset/en-us/products/storage/industry-market/h18564-powerprotect-data-manager-deployment-best-practice-wp.pdf
- How to Integrate PowerStore with PowerProtect Data Domain for Data Backup | Dell US
- For Kubernetes application consistency with PowerProtect Data Manager: https://dl.dell.com/content/manual45442344-powerprotect-data-manager-19-17-kubernetes-user-guide.pdf?language=en-us
- PPDM app consistency blog - https://infohub.delltechnologies.com/en-us/p/kubernetes-application-consistency-with-powerprotect-data-manager/
- Deploying PostgreSQL on Dell PowerFlex | Dell Technologies Info Hub

9.2 Backup and Restore – Milvus Database

Milvus provides robust backup and restore features to ensure data integrity and availability.

Backup Process:

- Metadata and Segments: Milvus Backup reads collection metadata and segments from the source Milvus instance to create a backup.
- Data Copying: It copies collection data from the root path of the source instance and saves it into the backup root path.

Restore Process:

- Step1: New Collection Creation: To restore from a backup, Milvus Backup creates a new collection in the target instance based on the collection metadata and segment information in the backup.
- Step2: Data Restoration: It then copies the backup data from the backup root path to the root path of the target instance.

Interfaces:

- CLI, API, and gRPC: Milvus Backup provides northbound interfaces such as CLI, API, and a gRPC-based Go module for flexible manipulation of the backup and restore processes.
- Minimal Performance Impact: The backup and restore processes have negligible impact on the performance of the Milvus cluster, allowing it to operate normally during these operations.

Usage:

- Command Line and API Server: The tool can be utilized through the command line or an API server, providing flexibility in how backups are managed.

For more detailed information, you can refer to the

- Milvus Backup Documentation: https://milvus.io/docs/milvus_backup_overview.md
- GitHub repository: https://github.com/zilliztech/milvus-backup

10. Conclusion

Robust server and storage components are critical for vector databases for several reasons: they ensure high performance by processing large volumes of high-dimensional data quickly and efficiently, they support seamless scalability to accommodate growing data and query load, and they provide high availability and fault tolerance to keep the database operational even during hardware failures. Additionally, they offer significant storage capacity to manage extensive datasets efficiently, they ensure data integrity and consistency with features like RAID, and they include advanced security measures to protect sensitive data from unauthorized access and breaches. In summary, robust server and storage components are the backbone of vector databases, ensuring they can handle large-scale, high-performance, and reliable data processing and storage needs.

10.1 Dell PowerEdge server for Vector Database

Dell's PowerEdge server portfolio is well-suited for vector database computing due to its robust performance, scalability, and reliability. Equipped with the latest Intel® Xeon® processors and high-capacity DDR4 memory, these servers handle the intensive computational demands of vector databases efficiently, ensuring fast data processing and query handling. The portfolio offers a range of configurations, including rack, tower, and modular infrastructure servers, allowing businesses to scale their infrastructure as their data needs grow. Additionally, features like RAID for data protection, high availability, and fault tolerance ensure continuous operation and data integrity, which are vital for maintaining the performance and availability of vector databases.

Furthermore, Dell servers support various in-server storage options, including all-flash configurations and hybrid tiered solutions, which is ideal for the large storage requirements of vector databases. Advanced management through the Dell OpenManage systems management portfolio simplifies server management with automation and remote access capabilities, reducing administrative overhead and ensuring optimal server performance. Dell’s innovative cooling technologies, such as Fresh Air 2.0, allow servers to operate at higher ambient temperatures, reducing cooling costs and improving energy efficiency. This combination of performance, scalability, reliability, and efficient storage makes Dell PowerEdge servers a strong choice for businesses looking to implement or upgrade their vector database infrastructure.

Some of the PowerEdge servers support unique optimizations to better match specific vector database requirements:

- The “xa” model is optimized for AI/ML environments. It delivers larger power supplies, high-performance cooling, and support for many GPUs to deliver the highest performance levels.
- The “standard” models are flexible enough to deliver enhanced virtualization support (with software-defined storage) or database performance (“in-memory” or traditional database) with the addition of high storage performance, large memory expansion, and increased core counts.
- The “xs” models deliver right-sized configurations for the most popular workloads, providing a balance of lower power consumption, a range of upgrade options, memory capacity, and performance as well as high-performance NVMe storage for demanding virtualization environments.
- The “xd2” model is designed for maximum storage capacity using large-form-factor spinning hard drives to deliver critical storage capacity for demanding environments such as video surveillance and object-based storage.

10.2 Dell Storage Solutions for Vector Database

Dell’s Power family of storage solutions is designed to meet the requirements of vector databases, ensuring optimal performance, scalability, and reliability.

PowerStore for transactional workloads like pgvector/PostgreSQL:
Dell PowerStore is designed to handle high-performance transactional workloads efficiently. It offers advanced data reduction capabilities, intelligent automation, and a scalable architecture. . This makes it ideal for databases like pgvector/PostgreSQL. PowerStore's ability to deliver consistent low-latency performance ensures that transactional operations are processed quickly and reliably.

PowerScale for scale-out workloads using file and S3 like Milvus:
Dell PowerScale provides a robust solution for scale-out workloads, particularly those involving substantial amounts of unstructured data. It supports both file and object storage interfaces, making it suitable for applications like Milvus that require scalable and flexible storage solutions. PowerScale's architecture allows for seamless scaling of capacity and performance, ensuring that large datasets can be managed efficiently.

11. Benchmark test setup Information.

11.1 Compute configuration information.

- HW Configuration
  - PE760xa, Dual CPUs (64 total cores), 768G ram, and 4 x Nvidia L40S
  - 2x 447GB nvme SSD in boss RAID1 configuration
  - 4x 446GB SATA SSD in perc RAID0 configuration
- Linux Install
  - Give Linux boot/application Raid1 virtual disk.
    - High reliability and read performance.
    - Linux system partition, root, and boot
  - Give Linux data to Raid0 virtual disk.
    - NVIDIA-SMI 550.78, Driver Version: 550.78, CUDA Version: 12.4
  - Applications
    - Give Docker all the system resources it wants.
      - No limit on ram and change the default storage directory to home (i.e., nvme)
    - Python 3.12.3
    - Milvus Standalone 2.4.4 (with and w/o GPU) running in docker.

11.2 Vector Database and Storage configuration information.

- **pgvector/PostgreSQL**

PostgreSQL was installed as standalone deployment in a VM in Vcenter/ESX environment.

- PostgreSQL Version used: 15.6-1.pgdg22.04+1
- One node standalone deployment in Ubuntu 22.o4TLS VM
- VM configuration
  - 8 CPU
  - 24 GB RAM

Leveraged PowerStore for VMWare Datastore store.

- Model: PowerStore 1000T

- **Milvus**

Milvus was installed on the Kubernetes cluster using the Operator and the default Cluster (etcd, Pulsar) and three nodes for each of Data, Index, and Query Nodes. Setup for Milvus is on a three node Kubernetes cluster hosted as virtual machines:

Kubernetes Server VM (x1)

- 16 CPU

- 32 GB RAM

Kubernetes Worker VM (x3)

- 32 CPU
- 32 GB RAM

Dell PowerScale F900 (OneFS v9.8.0.1) used for Milvus Storage for three node cluster, using the S3 interface.

The following is the Milvus cluster definition file:

```
# This is a sample to deploy a milvus cluster in milvus-operator's default
configurations.
apiVersion: milvus.io/v1beta1
kind: Milvus
metadata:
  name: milvus-s3
  labels:
    app: milvus
spec:
  mode: cluster
  dependencies: {}
  components: {}
  config:
    minio:
      bucketName: milvus-s3
      useSSL: false
  dependencies:
    storage:
      external: true
      type: S3    # MinIO | S3
      endpoint: ioflash-s3:9020
      secretRef: "milvus-s3-secret"
  components:
    dataNode:
      replicas: 3
    indexNode:
      replicas: 3
    queryNode:
      replicas: 3
```

12. Vector Database Use Cases

Vector databases have several use-cases that go beyond RAG:

1. Real-Time Recommendation Systems: In applications like e-commerce or streaming services, low latency is crucial for providing personalized recommendations. Vector databases can efficiently handle large volumes of user requests, process queries quickly, and generate relevant recommendations in real-time.

2. Fraud Detection: Financial institutions can use vector databases to analyze transaction patterns and detect anomalies. The ability to process high-dimensional data quickly helps in identifying fraudulent activities in real-time, ensuring timely intervention.

3. Image and Video Search: For platforms that rely on visual content, such as social media or digital asset management systems, vector databases can manage and query large datasets of images and videos. This enables efficient similarity searches and content retrieval based on visual features.

4. Natural Language Processing (NLP): In applications like chatbots or virtual assistants, vector databases can store and query large volumes of text data. It supports tasks such as semantic search, sentiment analysis, and language translation, providing accurate and timely responses.

5. Healthcare Data Analysis: Vector databases can be used to manage and analyze complex medical data, including patient records, imaging data, and genomic information. This facilitates advanced diagnostics, personalized treatment plans, and medical research.

6. IoT Data Management: In IoT applications, vector databases can handle the continuous influx of sensor data. They support real-time data ingestion, processing, and querying, which is essential for monitoring and controlling IoT devices.

13. Bibliography

Accelerating Vector Search: Using GPU-Powered Indexes with RAPIDS RAFT | NVIDIA Technical Blog